TECHNICAL REPORT SERIES
N°215 April 1986

Software monitoring of UNIX systems

A. Montasar-Kohsari
I. Mitrani
T. Betteridge

Abstract

The structure and application of a software package for monitoring the performance of a Unix system at the kernel call level is described. That package, referred to as MOLLUSC, can be used both in an isolated Unix system and in a distributed environment such as the one provided by the Newcastle Connection. It enables the user to collect and analyse statistics about the local and/or remote kernel calls issued by one or more programs, running in parallel or in sequence. The results of several monitoring experiments where MOLLUSC was used to study the usage distribution of various kernel calls and their performance under different conditions are reported.

Series Editor: M. J. Elphick

Printed and published by the University of Newcastle upon Tyne,
Computing Laboratory, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

MONTASER-KOHSARI, Afzal

Software monitoring of UNIX systems. [By] A. Montasar-Kohsari, I. Mitrani and T. Betteridge.

Newcastle upon Tyne: University of Newcastle upon Tyne, Computing Laboratory, 1986.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 215.)

Added entries

MITRANI, Israel.
UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 215.

Abstract

The structure and application of a software package for monitoring the performance of a Unix system at the kernel call level is described. That package, referred to as MOLLUSC, can be used both in an isolated Unix system and in a distributed environment such as the one provided by the Newcastle Connection. It enables the user to collect and analyse statistics about the local and/or remote kernel calls issued by one or more programs, running in parallel or in sequence. The results of several monitoring experiments where MOLLUSC was used to study the usage distribution of various kernel calls and their performance under different conditions are reported.

About the author

Dr. Montaser-Kohsari is a Research Associate at the Computing Laboratory, University of Newcastle upon Tyne.

Dr. I. Mitrani is a Lecturer at the Computing Laboratory, University of Newcastle upon Tyne.

Dr. T. Betteridge is a Systems Programmer at the Computing Laboratory, University of Newcastle upon Tyne.

Suggested keywords

DISTRIBUTED SYSTEMS
PERFORMANCE EVALUATION
SOFTWARE MONITORING
UNIX

Suggested classmarks (primary classmark underlined)

Dewey (18th): 001.64404 001.6425

U. D. C. 519.687 681.322.06

Software Monitoring of UNIX† Systems

A. Montaser-Kohsari, I. Mitrani and T. Betteridge

Computing Laboratory, University of Newcastle upon Tyne

ABSTRACT

The structure and application of a software package for monitoring the performance of a Unix system at the kernel call level is described. That package, referred to as MOLLUSC, can be used both in an isolated Unix system and in a distributed environment such as the one provided by the Newcastle Connection. It enables the user to collect and analyse statistics about the local and/or remote kernel calls issued by one or more programs, running in parallel or in sequence. The results of several monitoring experiments where MOLLUSC was used to study the usage distribution of various kernel calls and their performance under different conditions are reported.

1. Introduction

It is always interesting, and often important, to find out how frequently and how quickly an operating system carries out its various tasks. In the case of UNIX, where all system services are requested and performed by means of system calls, the desired information can be obtained by monitoring those calls and collecting relevant statistics about them. This is the purpose of the package described here.

MOLLUSC (Monitor Of Local and Long-distance Unix System Calls) is a software layer which sits, conceptually, between the user programs and the Unix kernel. A user wishing to monitor the execution of a program may specify an interest in some, or all system calls. Every invocation of one of those calls is then intercepted by MOLLUSC, which outputs a record describing that invocation into a trace file (figure 1). The statistics collected in the trace file are analysed separately.

Although it can be used on its own, in an isolated Unix system, MOLLUSC was developed primarily as an adjunct to the Newcastle Connection. The latter makes several Unix systems, connected by some communication channel, appear to the users as a single Unix system (see, for instance [1,3]). In such an environment, MOLLUSC can sit between the user programs and the Newcastle Connection on one (or more) of the constituent systems. The Connection then becomes, as far as MOLLUSC is concerned, part of the kernel of the global Unix system. System calls issued in the home system can thus be monitored, regardless of whether they are executed locally or remotely. This mode of operation is illustrated in figure 2.

---

† UNIX is a Trademark of Bell Laboratories.

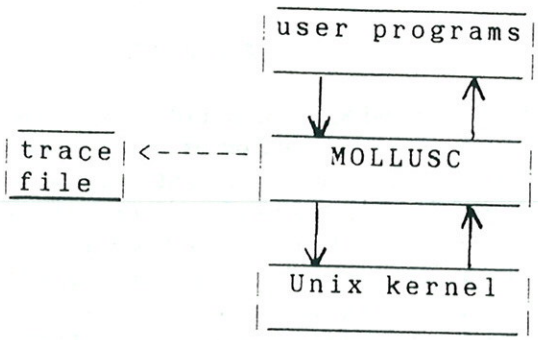

Figure 1.

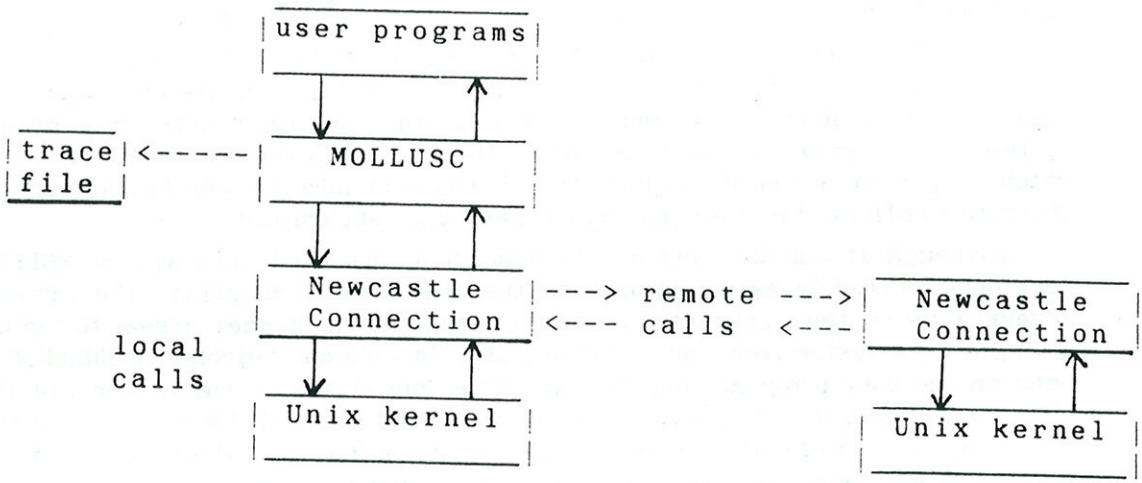

Figure 2.

In the existing implementations, MOLLUSC is always in the user space, whereas the Newcastle Connection may or may not in fact be inside the kernel.

2. Program structure

The existing implementation of MOLLUSC is for the PNX version of Unix supplied with the PERQ workstation. That version contains 48 system calls. Hence, the main part of MOLLUSC consists of 48 routines, each corresponding to a Unix system call and having the latter's name as an entry point. These routines are integrated into the C-library, after renaming the entry points of the real system calls (or their Newcastle Connection equivalents). The effect of this is that, whenever a user program which has been linked with the C-library makes a system call, the corresponding MOLLUSC routine is invoked. The same approach applies to other Unix systems, although different sets of routines may be required. An implementation for Unix 4.2 is being developed at present.

From now on, when we refer to a 'real system call', we shall mean either the original Unix entry point of that name, or the Newcastle Connection one, if it exists. This will obviate the need to distinguish between systems where the Newcastle Connection has been installed, and those where it has not. The MOLLUSC routines will be referred to as 'pseudo system calls'.

Each pseudo system call can operate in one of two possible modes: 'passive' and 'active'. In the passive mode, its only action is to invoke the corresponding real system call and return the result to the user program. In the active mode, it does the same but also assembles and outputs a record containing a few items of information about that invocation. The switching between the two modes is controlled by a string of 48 characters which must be present in the environment of the user process. The pseudo calls are numbered from 1 to 48. Whenever the i'th call is invoked, it examines the i'th character in the control string and operates in passive or active mode depending on whether that character is 'N' or 'Y' respectively.

Most of the records written into the trace file contain five items. These are:

(i) The identifying number of the system call;

(ii) The time of day when the user made the call (in milliseconds);

(iii) The time it took to execute the real system call (in milliseconds);

(iv) The return code of the real system call;

(v) The file descriptor of the file manipulated by the call, if any (this applies only to calls such as 'read', 'write', 'ioctl', etc).

Note, that item (iii) refers to the elapsed time, i.e. the interval between making the real system call and getting the result back. The reason for including item (iv) is partly in order to know whether the call was successful or not, but also because the return code sometimes provides useful information (e.g. the number of bytes read or written, in the case of read or write calls).

The exceptions to the above pattern are the 'exec' and 'exit' calls which, when successful, do not return to the caller. In those cases, the records do not contain items (iii) and (iv). However, the record of the 'exit' call contains the process identifier of the exiting process.

Figure 3 shows an example of a short trace file. The program being monitored is the Unix command 'cat'.

```
19 14318183    467      0  1
31 14318650    250      3
19 14318916      0      0  3
24 14318916      0     -1  3
33 14318916    100     58  3
24 14319016      0     -1  3
33 14319033      0      0  3
 6 14319033      0      0  3
 6 14319033      0      0  0
46 14319050    183     58  1
 6 14319233     17      0  1
 6 14319250      0      0  2
48 14319250            143  0
```

Figure 3.

It should be clear from this illustration that the 'raw' information in the trace file is not in an easily useable form, especially when the file is long. A program is therefore provided that reads the trace file and produces summary statistics. These include the names, as well as the identifying numbers of the system calls used by the user program, the number of times each of those calls was invoked, the sum of its execution times, the total execution time of all the calls (this last quantity is referred to as the 'system time' of the user program), the fraction of that total occupied by each call (there is also a graphic representation of those fractions), the numbers of bytes read and written, and the number of successful 'exec' invocations. The summary file corresponding to the trace in figure 3 is shown in figure 4.

It should be emphasised that our definition of call execution time, and hence that of system time, includes all kinds of delays imposed on or by the kernel. These may be caused by disk arm movements, roll-in/roll-out of processes, transmissions of messages along communication channels, etc. Another point to bear in mind is that, if the user program does a 'fork', and both father and son continue to run in parallel, all the system calls that they invoke appear in the trace file. It may therefore happen that, in the summary statistics, the program's system time is larger than its elapsed time (the call execution times of the two processes may overlap in real time). However, any 'wait' calls issued by the user program are not included in its system time.

Before we go on to report the results of some monitoring experiments, a few words are in order about the overheads of MOLLUSC. First the run-time ones.

Much effort has been taken to make the MOLLUSC routines as economical as possible (for instance, the output records are buffered in memory, up to 512 of them at a time, in between writes to the trace file). Nevertheless, monitoring a program increases its elapsed time by about 20%. However, the processing that those routines carry out (such as examining the environment, preparing the output records, etc.) is

```
code   name    noinv     ttime          ave          sd       fract
  6    close     4         0.0          0.0         0.0       0.000
 19    fstat     2       467.0        233.5       233.5      46.700
 24    ioctl     2         0.0          0.0         0.0       0.000
 31     open     1       250.0        250.0         0.0      25.000
 33     read     2        83.0         41.5        41.5       8.300
 46    write     1       200.0        200.0         0.0      20.000
 48    _exit     1         0.0          0.0         0.0       0.000

start time:15600883

bytes written: total=          58 ave=       58.0 sd=         0.0
bytes read:    total=          58 ave=       29.0 sd=        41.0

the no of successful exec=0

total system time=      1.000
total usr time    =     0.083
the elapsed time  =     1.083

no of system call used = 7
close |
ioctl |
_exit |
read  |*********
write |********************
open  |*************************
fstat |************************************************
```

Figure 4.

done outside the intervals occupied by the invocations of the real system calls. The timing of the latter is, therefore, accurate.

Next, space. Since the user program is linked with the monitoring routines that it invokes, its size may be increased considerably. That increase is in addition to the one that may be entailed by linking with the Newcastle Connection. Figure 5 shows the sizes of three Unix programs - cat, cp and sh - by themselves, linked with the Newcastle Connection only, and linked with the Newcastle Connection and MOLLUSC.

Of course, in systems where the Newcastle Connection is part of the kernel, its space overheads are largely eliminated. This has not yet been done with MOLLUSC. However, there are other ways of reducing significantly its space requirements and

	cat	cp	sh
alone	7168	6144	20480
with N.C.	33320	30312	64552
with N.C. and MOLLUSC	40656	39584	80664

Figure 5.

these will be explored in future releases.

3. Monitoring experiments.

One of the first questions that arise in connection with the performance of Unix concerns the usage distribution of the different kernel calls. It turns out that the latter is extremely skewed. There is a group of five or six calls which, between them, occupy more than 95% of all system time. Two of those calls - read and write - on their own account for over 80% of the system time.

This situation is illustrated in figure 6, (a), (b) and (c). A 20-minute session involving a variety of tasks was monitored and the results were saved in a common trace file. The summary statistics corresponding to that file are shown in 6 (a). The fractions of the total system time attributable to the different calls (last column in 6 (a)) are plotted in 6 (b). Note that the 'wait' call is included in the plot, but not in the total system time. The dominant calls are seen to be read, write, stat, creat and open.

The picture displayed by these statistics is quite typical. Admittedly, small variations are observed in different applications. Some of the calls may change their relative fractions of system time, but not to any great extent. In particular, the preeminent position of read and write appears to be universal.

The commands which were executed during the above session are listed in 6 (c).

A feature of the statistics which is worth pointing out, is the variability of call execution times. In most cases where a call has been invoked more than once or twice, the standard deviation of the execution times is greater than the mean. This variability will be a recurring theme throughout these discussions.

The next series of experiments was designed to evaluate the performance of a task involving two Unix machines communicating over an Ethernet by means of the Newcastle Connection. The task chosen was the copying of a reasonably large file (approximately 170K bytes) from one PERQ station to another, or to a different place in the same station. The two machines are referred to as Amble and Tweed, respectively.

There are four possibilities for the machines where the source and destination files are located. All four were examined. Accordingly, the results are displayed in the form of a two by two matrix, where the rows are identified by the location of the source file and the columns by that of the destination (see figure 7(a)). In all cases,

code	name	noinv	ttime	ave	sd	fract
0	access	246	10518.0	42.8	25.0	0.999
2	chdir	5	266.0	53.2	58.8	0.025
4	chown	3	66.0	22.0	7.8	0.006
6	close	791	17957.0	22.7	75.6	1.705
7	creat	226	32208.0	142.5	387.4	3.059
8	dup	3	16.0	5.3	7.5	0.002
12	excrve	47	0.0	0.0	0.0	0.000
15	execv	2	600.0	300.0	300.0	0.057
16	execve	2	517.0	258.5	258.5	0.049
18	fork	98	2189.0	22.3	18.4	0.208
19	fstat	134	3385.0	25.3	108.6	0.321
20	ftime	21	34.0	1.6	5.0	0.003
21	getgid	1	0.0	0.0	0.0	0.000
22	getpid	93	99.0	1.1	4.1	0.009
23	getuid	5	0.0	0.0	0.0	0.000
24	ioctl	1346	5077.0	3.8	38.8	0.482
26	link	3	216.0	72.0	31.1	0.021
27	lseek	87	66.0	0.8	3.5	0.006
28	mknod	1	200.0	200.0	0.0	0.019
31	open	441	42409.0	96.2	130.4	4.027
33	read	5056	419180.0	82.9	102.0	39.809
35	setuid	1	0.0	0.0	0.0	0.000
36	signal	20	0.0	0.0	0.0	0.000
37	stat	954	93645.0	98.2	164.2	8.893
40	time	7	0.0	0.0	0.0	0.000
41	umask	2	0.0	0.0	0.0	0.000
43	unlink	232	20960.0	90.3	124.5	1.991
46	write	5537	403378.0	72.9	246.4	38.308
48	_exit	43	0.0	0.0	0.0	0.000
45	wait	49	93617.0	1910.6	460.9	8.165

```
start time:85593966
bytes written: total=    2044316 ave=      369.2 sd=      225.0
bytes read:    total=    2185384 ave=      432.2 sd=      483.3
the no of successful exec=49
total system time=  1052.986
total usr time    =  1423.131
the elapsed time =  2476.117
no of system call used = 30
```

Figure 6(a)

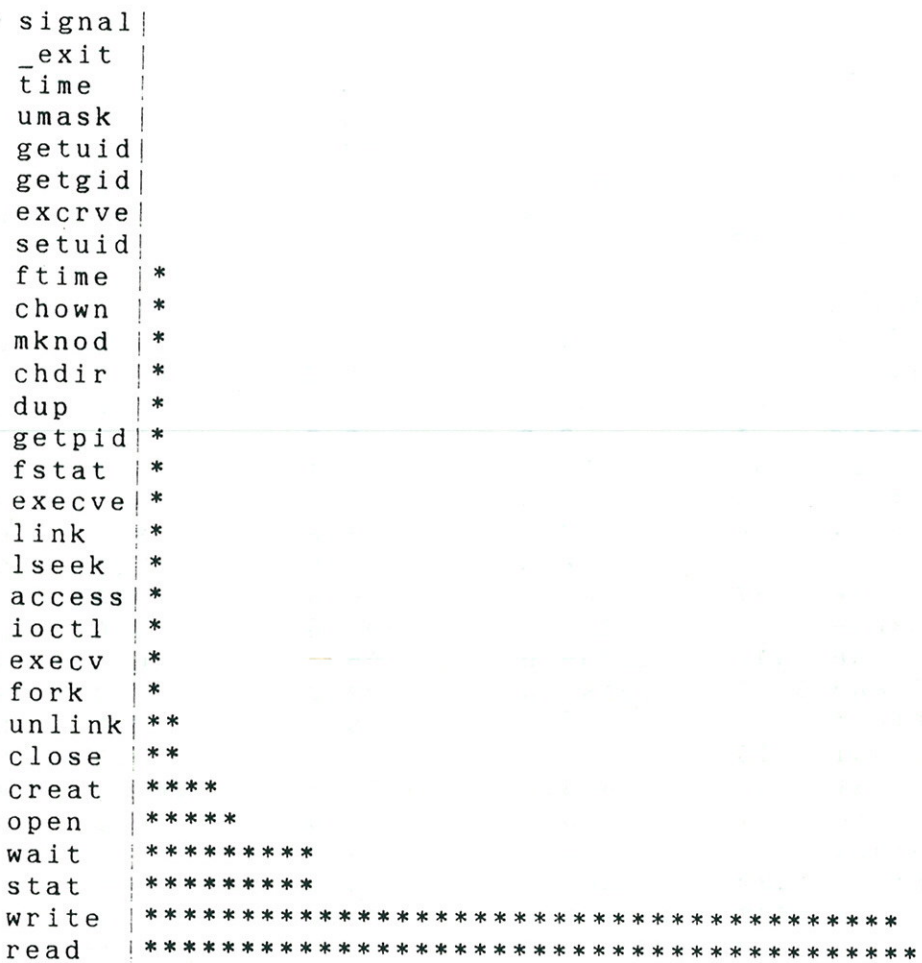
signal
_exit
time
umask
getuid
getgid
excrve
setuid
ftime
chown
mknod
chdir
dup
getpid
fstat
execve
link
lseek
access
ioctl
execv
fork
unlink
close
creat
open
wait
stat
write
read

Figure 6(b)

```
cptree /usr/lib alaki
rm -r alaki
cat infor.c
chmod a+x infor
chown nc infor
chmod a+x test
chown nc test
cat acce.c
cmp testmulti.c sysmore.c
mkdir alaki
cptree -p /fd/Scsu alaki
rm alaki/*
rmdir alaki
echo hello world
mv test test1
cp sysmore.c sys.c
who
pwd
ls -l
sh /usr/goli/ssp
grep print sysmore.c
mv sys.c sys1.c
cp sys1.c sys2.c
rm sys1.c
ps ax
file sysmore.c
stty
sleep 10
time sh ssp
ls /
stty
cat /fd/SMncsys/*>test
rmdir alaki
rm test
cat /fd/SMncsys/*
cptree /fd/SMncsys alaki
rm alaki/*
rmdir alaki
echo that is all
ps ax
bye
cat unixp1 unixi>unixp1
cp unixp1 unix1
```

Fiqure 6(c)

MOLLUSC resided on Amble and the copy command was issued there. That experiment was performed at night, so as to minimise the interferences on the network as far as possible. No other processes were active at either machine.

As well as the system time (ST) and the elapsed time (ET), figure 7(a) shows the average time per invocation, and the total time, for the read, write, open and stat calls. These are the only calls that take up significant amounts of system time in this task.

The information in each square of the table is the result of ten repeated runs of the same command. The latter yield point estimates and 90% confidence intervals for the corresponding performance measures. In deriving those confidence intervals it was assumed that the ten observations are independent, normally distributed random variables with the same mean and standard deviation. A bar chart representation of the ST and ET values in this table is shown in figure 7(b).

Compare first the copy from Amble to Amble with that from Amble to Tweed. In the former case all operations are local, whereas in the latter the reads are local and the writes (and stats) are remote. One might expect, a priori, that the second copy would take longer than the first. In fact we observe a slight reduction in both the system and the elapsed times. A closer look at the individual operations reveals that, although the write time has indeed increased, there has been a remarkable reduction in the read time. A likely explanation for this is that the copy from Amble to Tweed involves less disk arm movement between consecutive reads than the one from Amble to Amble.

Similar observations can be made when comparing the Amble -> Amble with the Tweed -> Amble operations. In the latter case the reads are remote and the writes are local. Again the read times are reduced, presumably for the above reason. It is unclear why the local writes should take longer to execute. The overall reduction in system and elapsed times is now even more pronounced.

The worst performance is observed in the Tweed -> Tweed case, where both files are on the same disk and all operations are remote. The amount of time necessary to do a single remote open is quite remarkable - 3.6 seconds!

Our first tentative conclusion is that the fact that a task requires communications over a network does not necessarily have an overwhelming influence on its performance. Other factors, such as the physical locations of files on the surface of a disk may be equally, or more important.

The same experiment was carried out twice during the day (in the morning and in the afternoon), when there was some likelihood of interferences on the network. the results are shown in figure 8. Only the entries Amble -> Tweed and Tweed -> Amble contain both sets of numbers. The differences observed between the night-time, morning and afternoon runs are less than 10% ; moreover, they are not always in the expected direction. It is likely, therefore, that the reasons for those differences have less to do with network interferences than, say, with other file and directory changes.

Since the search for 'natural' interferences proved inconclusive, it was decided to introduce artificially, and in a controlled way, some competition for the available resources. This took two forms: another process running in parallel on one of the machines and another stream of packets being transmitted in parallel on the Ethernet.

Figure 9 shows the statistics when the copy task competes with a process (a clock) running in parallel on Tweed. Predictably, there is a significant deterioration

	Amble	Tweed
Amble	ST = 27560±1533 ET = 32170±1534 Single Total read 69.16±4.18 23376±1411 write 11.25±0.52 3791± 175 open 186.7± 4.0 187± 4 stat 0 0	ST = 27151± 525 ET = 31823± 539 Single Total read 19.60±1.22 6625± 412 write 49.52±0.47 16688± 158 open 183± 0.3 183± 0 stat 1674± 52.3 3348± 105
Tweed	ST = 24877± 508 ET = 29553± 452 Single Total read 42.84±1.07 14480± 361 write 19.36±1.56 6524± 526 open 3408± 102 3408± 102 stat 15.1±11.7 30± 23	ST = 36051± 468 ET = 39987± 478 Single Total read 53.47±1.25 18073± 422 write 41.48±1.31 13979± 440 open 3598± 49.1 3598± 49 stat 49.15±1.54 98.3± 3

Figure 7(a) cp, night-time (no other interference).

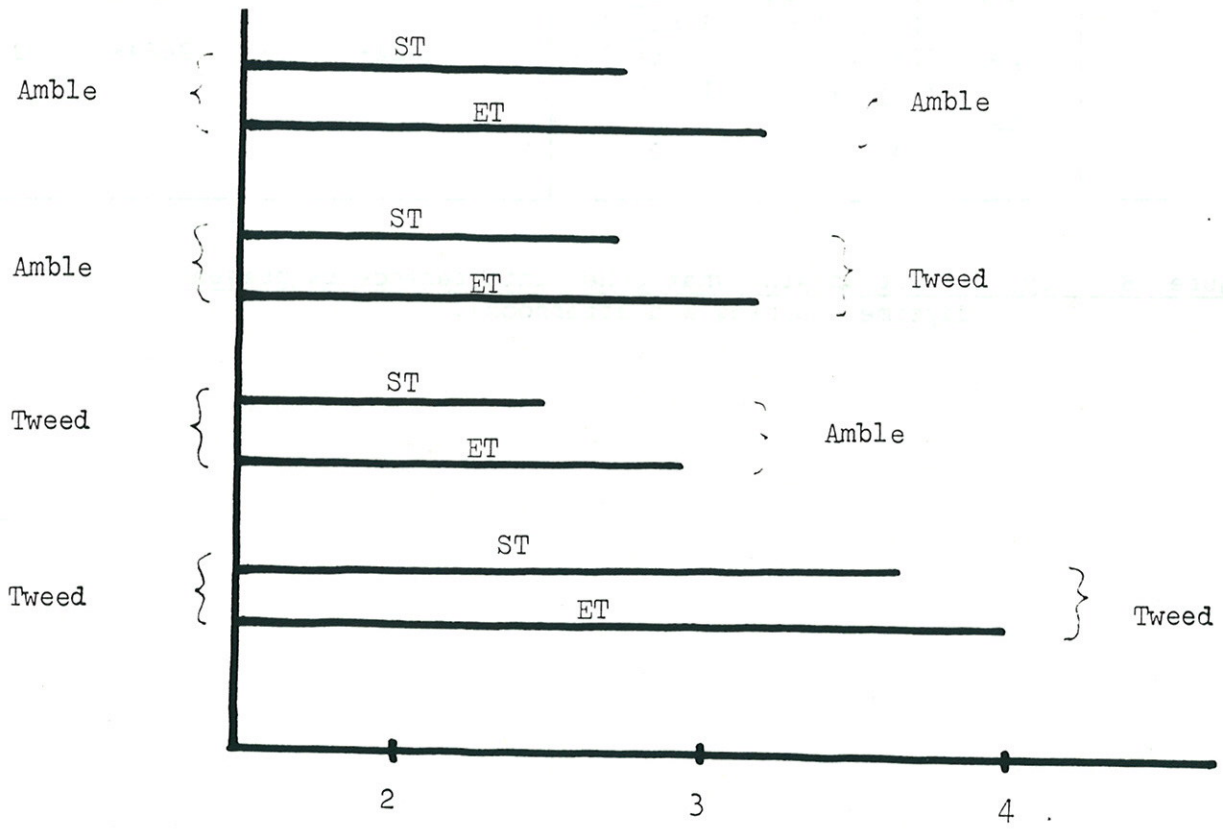

Figure 7(b): Average ST and ET values (in seconds)

	Amble	Tweed
Amble	ST = 27560±1533 ET = 32170±1534	ST = 27210± 461 25992± 986 ET = 31638± 524 30605± 985
	Single / Total	Single / Total
	read 69.16±4.18 / 23376±1411	read 21.08±0.91 / 7125± 309 19.60±0.94 / 6625± 319
	write 11.25±0.52 / 3791± 175	write 47.82±0.80 / 16115± 268 51.93±2.07 / 17500± 697
	open 186.7± 4.0 / 187± 4	open 190± 4.9 / 190± 5 190± 4.9 / 190± 5
	stat 0 / 0	stat 1723± 89 / 3447± 178 728± 6.7 / 1455± 13
Tweed	ST = 24889± 407 23277± 447 ET = 29277± 480 28230± 412	ST = 34397± 367 ET = 38590± 394
	Single / Total	Single / Total
	read 45.26±0.78 / 15298± 263 46.39±0.83 / 15680± 279	read 48.17±1.23 / 16281± 415
	write 17.83±0.85 / 6009± 285 17.15±1.02 / 5780± 343	write 41.43±0.39 / 13962± 132
	open 3278± 379 / 3278± 379 1734± 44 / 1734± 44	open 3624± 9.3 / 3624± 9.3
	stat 0 / 0 0 / 0	stat 0 / 0

Figure 8. cp, without knowing what other interference is present, daytime (morning and afternoon).

	Amble	Tweed
Amble	ST = 27745± 870 ET = 32698± 869	ST = 35862±5145 ET = 40495±5072
	Single / Total	Single / Total
	read 68.74±3.16 / 23234±1067	read 18.95±0.68 / 6405± 230
	write 11.22±0.40 / 3781± 134	write 74.2±14.9 / 24992±5026
	open 203.3±17.3 / 203± 17	open 213± 41.3 / 213± 41
	stat 5.85±2.32 / 11.7± 4.6	stat 1904± 116 / 3808± 231
Tweed	ST = 33165±2454 ET = 37838±2501	ST = 38179±1755 ET = 42380±1740
	Single / Total	Single / Total
	read 68.49±7.28 / 23150±2460	read 52.81±1.38 / 17850± 467
	write 18.00±1.15 / 6066± 386	write 47.65±4.89 / 16058±1647
	open 3682± 340 / 3682± 340	open 3923± 263 / 3923± 263
	stat 8.4± 2.7 / 17± 5.5	stat 50± 2.5 / 100± 5.0

Figure 9. cp, with a clock running on Tweed, daytime (other interference unknown)

of performance in all three cases involving Tweed. This is directly attributable to an increase in the execution times of the remote calls.

A notable feature of figure 9 is that the confidence intervals of the system and elapsed times, in the cases involving Tweed, are much larger than before. This indicates that the extra process not only makes the average performance worse, but also increases the variability of individual execution times. Of course, the wider the confidence intervals, the less accurate are the point estimates.

The other form of interference that was introduced was a second communication task (listing a large directory from a machine called Wansbeck to one called Coquet) proceeding in parallel with the copy task. The resulting statistics are displayed in figure 10. The competition for the network appears, in this case, to have a smaller influence on performance than that for the machine. However, there is a similar increase in the variability of execution times, accompanied by a corresponding widening of the confidence intervals.

A similar set of experiments were performed with a bigger and more complicated task. This consisted of concatenating two quite large files (total size approximately 340K bytes) and putting the result into a third file. Again, the two machines Amble and Tweed were used. There are now 8 possibilities for the locations of the three files. It is convenient to display the results in the form of a (4x2) matrix, where the rows are identified by the locations of the two source files and the columns - by the location of the destination file. In all cases, MOLLUSC resides on Amble and the cat command is issued there.

The task was performed on its own (at night), in competition with a process running in parallel on Tweed, and in competition with another pair of communicating machines using the Ethernet at the same time. The results are shown in figures 11, 12 and 13, respectively.

The aspect of these last figures is somewhat different and, perhaps, more predictable than that of figures 7 - 10. The phenomenon of a remote task taking less time than a local one is no longer observed. This is probably because, in all configurations, there are at least two files on one disk. The differences in performance are more pronounced. For example, the system time in the worst case (all three files on Tweed) is larger than that in the best case (all three files on Amble) by a factor of more than two.

The comparison between the effects of the two different types of interference is not as clear-cut as before. We observe that when the destination file is on the local machine (Amble), the competing process has a bigger influence on performance than the communication task. The position is reversed when the destination task is on the remote machine. We do not pretend to understand fully the reason for this. However, the following general property of CSMA networks may supply a partial explanation: Their performance is a non-linear, convex function of the load placed on them. That is, the higher the load on the network, the faster its performance deteriorates when that load is increased. It may be that the traffic generated by the remote writing of the big destination file is sufficiently heavy to make the influence of the extra communication task more noticeable.

	Amble	Tweed
Amble		ST = 28757±1497 ET = 33938±1519 Single Total read 19.68±0.61 6652± 207 write 53.58±3.70 18056±1247 open 185± 3.0 185± 3 stat 1772± 107.0 3543± 214
Tweed	ST = 25624±1002 ET = 30115±1101 Single Total read 47.91±2.99 16194±1009 write 16.45±1.11 5544± 374 open 3422± 131 3422± 131 stat 8.5± 2.8 17± 5.7	ST = 39645±1874 ET = 43653±1847 Single Total read 58.73±4.92 19851±1663 write 46.27±1.81 15593± 610 open 3554± 102 3554± 102 stat 50± 2.5 100± 5.0

Figure 10. cp, with ls -l from Wansbeck to Coquet, night-time (no other interference).

	Amble		Tweed	
cat Amble - Amble	ST = 29981±1504 ET = 74358±1364		ST = 43881± 556 ET = 79862± 557	
	Single	Total	Single	Total
read	28.35±2.00	19165±1350	17.68±0.82	11952± 554
write	12.89±0.40	8688± 268	44.87±0.67	30242± 454
ioctl	1.07±0.10	723± 66	0.87±0.10	588± 68
open	221± 12	442± 24	123± 18	246± 35
cat Tweed - Amble	ST = 44325±1661 ET = 86090±1491		ST = 54109± 565 ET = 89972± 647	
	Single	Total	Single	Total
read	32.45±1.88	21936±1270	26.38±0.43	17833± 291
write	16.68±0.64	11242± 429	43.28±0.53	29171± 356
ioctl	9.78±0.51	6611± 343	8.80±0.11	5949± 74
open	1808± 70	3617± 140	187± 3.4	373± 6.8
cat Amble - Tweed	ST = 46924±1313 ET = 88192±1207		ST = 56218± 484 ET = 91667± 552	
	Single	Total	Single	Total
read	35.25±1.65	23829±1115	26.38±0.43	17833± 288
write	17.94±0.92	12092± 619	45.14±0.80	30424± 537
ioctl	9.42±0.09	6368± 62	9.52±0.13	6436± 91
open	1885± 55	3770± 109	287± 23	575± 46
cat Tweed - Tweed	ST = 52586± 347 ET = 89047± 759		ST = 72713± 902 ET =107544± 842	
	Single	Total	Single	Total
read	39.90±0.35	26972± 238	42.95±0.48	29034± 327
write	15.81±0.44	10656± 297	44.95±1.04	30296± 702
ioctl	17.66±0.55	11938± 370	18.08±0.36	12222± 246
open	982± 162	1963± 324	501± 31	1001± 61

Figure 11. Working on Amble, night time.

	Amble			Tweed		
cat Amble - Amble	ST = 29981±1504 ET = 74358±1364			ST = 45679±529 ET = 82802±647		
		Single	Total		Single	Total
	read	28.35±2.00	19165±1350	read	17.71±0.27	11972±180
	write	12.89±0.40	8688± 268	write	47.79±0.74	32306±500
cat Tweed - Amble	ST = 44666± 947 ET = 86080± 707			ST = 59400±296 ET = 95438±319		
		Single	Total		Single	Total
	read	33.22±1.22	22457± 822	read	28.54±0.44	19293±297
	write	16.11±0.59	10890± 397	write	43.32±0.54	29284±368
cat Amble - Tweed	ST = 47313±1221 ET = 99000±1401			ST = 58573±730 ET = 95248±732		
		Single	Total		Single	Total
	read	35.82±1.64	24214±1111	read	26.82±0.35	18130±239
	write	17.36±0.67	11735± 450	write	47.67±0.60	32225±406
cat Tweed - Tweed	ST = 62221± 608 ET = 99556± 477			ST = 73682±843 ET =110135±802		
		Single	Total		Single	Total
	read	43.34±0.75	29298± 508	read	37.56±0.83	25391±560
	write	22.74±0.47	15372± 314	write	49.67±0.38	33577±256

Figure 12. Working on Amble, with clock on Tweed.

	Amble		Tweed	
cat Amble - Amble	ST = 29981±1504 ET = 74358±1364		ST = 49458±1136 ET = 86127±1200	
	Single	Total	Single	Total
read	28.35±2.00	19165±1350	17.62±0.37	11911± 249
write	12.89±0.40	8688± 268	53.46±1.72	36032±1162
cat Tweed - Amble	ST = 44523±1439 ET = 85457±1408		ST = 63916±1854 ET = 99642±1923	
	Single	Total	Single	Total
read	34.21±1.57	23126±1059	28.61±1.40	19340± 945
write	15.64±0.97	10541± 654	54.14±2.12	36490±1432
cat Amble - Tweed	ST = 46093±1053 ET = 86130±1059		ST = 65102± 541 ET =101325±1358	
	Single	Total	Single	Total
read	37.58±1.51	25404±1023	30.12±1.12	20361±755
write	14.28±0.64	9625± 429	56.94±0.77	38377±521
cat Tweed - Tweed	ST = 57955±1335 ET = 94182±1363		ST = 76771±1547 ET =111845±1402	
	Single	Total	Single	Total
read	45.19±1.94	30548±1311	41.79±1.73	28250±1166
write	14.80±0.76	9975± 513	50.60±1.90	34104±1279

Figure 13. Working on Amble, with "ls -l" running on Wansbeck and Coquet.

Conclusions.

The software package described here is a convenient and useful tool for studying the performance of a Unix system. Observing the behaviour of a process, or processes, at the kernel call level can provide valuable information which would not otherwise be available. That information could also help in finding the best way to improve the performance of a given system.

The experimental results reported here are, of course, dependent on the particular systems and machines that were used. However, we believe that certain qualitative features of those results, such as the predominance of a small group of system calls and the large variability of call execution times, are common to most Unix systems.

Perhaps the most interesting aspect of these experiments has to do with the questions that they raise, rather than with those that they answer. Of particular interest is the extent to which different tasks (even functionally independent ones) interact in a multiprocessing/distributed system environment. This is a matter which deserves further, and more detailed study.

References.

[1] Brownbridge, D.R., Marshall, L.F. and Randell, B., "The Newcastle Connection - or UNIXes of the World Unite!", Software Practice and Experience, 12(12), pp. 1147-1162 (1982).

[2] Linton, A. and Panzieri, F. "A Communication System Supporting Large Datagrams on a Local Area Network", Report 191, Computing Laboratory, University of Newcastle upon Tyne (1984).

[3] Randell, B. "The Newcastle Connection: A Software Subsystem for Constructing Distributed UNIX Systems", Report 194, Computing Laboratory, University of Newcastle upon Tyne (1984).

[4] Ritchie, D.M. and Thompson, K., "The UNIX Time Sharing System", Comm. ACM, 17(7), pp. 365-375 (1974).